Communication Membrane Systems with Active Symports

Chapter 8

Communication Membrane Systems with Active Symports

Abstract
We consider membrane systems where the generation/transformation of objects can take place only if it is linked to communication rules.

More specifically, all the rules move objects through membranes and, moreover, the membranes *can* modify the objects as they pass through. The intuitive interpretation of such rules is that a multiset of objects can move from a region to an adjacent one, and moreover objects can engage into (biochemical) reactions while passing through (are in "contact" with) a membrane. Therefore such "twofold" rules are called symport-rewriting (in short, *sr*) rules, where symport refers to a coordinated passage of a "team" of molecules through a membrane.

In this chapter we investigate the influence of the form of sr rules on the power of membrane systems that employ them (sometime in combination with simple antiport rules which allow a synchronized exchange, through a membrane, of two molecules residing in two adjacent regions). A typical restriction on the form of an sr rule requires that the passage described by the rule is such that the sort of exiting molecules is a subset of the sort of entering molecules (however the multiplicities of sorts do not have to be related).

We also compare the sequential passage mode with the maximally parallel passage mode.

8.1 Introduction

Membrane computing, introduced in [20], is a computational model inspired by the functioning of membranes in living cells. The biological membranes within a cell divide the cell in a number of compartments (regions). This is the basic feature of this model with each region containing its own set of evolution rules,

where each rule prescribes both the transformation and generation of molecules as well as the transport of molecules through membranes. In this way each evolution rule has both a rewriting and a communication component.

An important class of membrane systems allows only communication, i.e., their rules prescribe only the passage of objects (molecules) through membranes. Such systems are called symport/antiport membrane systems where both "symport" and "antiport" refer to types of rules that allow for a synchronized passage of molecules through a membrane. For symport rules this passage is unidirectional (a multiset of molecules is passing through a membrane together), while for antiport rules this passage is bidirectional (a passage of a multiset of molecules in one direction is synchronized with a passage of molecules in another direction through the same membrane).

In this chapter we enrich symport rules by coupling them with a generative component: a multiset of molecules passing a membrane synchronously in the unidirectional fashion can be changed to a different multiset. Such rules are called *symport-rewriting rules*, or *sr rules* for short – they are biologically motivated, as the biological membranes do not only allow for the passage of molecules, but can also change them *during* a passage. Membrane systems using sr rules and antiport rules are called *communication membrane systems with active symports*, or CAS P *systems* for short.

In this chapter we study the influence of the form of sr rules (sometimes in combination with antiport rules) on the generative power of resulting membrane systems. We also study the influence of the communication mode (sequential versus maximally parallel) on the generative power. In sequential mode, at any given time, at most one sr rule can be active for any given membrane, while (as usual) in maximally parallel mode an application of the rules is such that no more rules can be applied to the objects that are not already involved in the passage through membranes.

The chapter is organized as follows. Section 8.2 recalls some basic notions from language theory – more specifically those of matrix grammars and register machines. Section 8.3 formally defines CAS P systems. Section 8.4 considers "alphabetically restricted" CAS P systems which can use only sr rules in which the type of every generated object (i.e., an object occurring at the right hand side) is already present at the "entrance to the membrane" (i.e., occurring at the left hand side). In Section 8.5 we consider CAS P systems operating in sequential mode. Section 8.6 considers CAS P systems that use only unary rules, i.e., the rules that describe the passage of one single molecule and allow this molecule to multiply during this passage (technically, this corresponds to the rules $a \to v$, where $v \in a^*$).

A natural restriction on communication can be formulated through the use of uni-directional membranes, i.e., membranes such that, for any symbol a, if a may cross a given membrane in one direction (as specified by one of the rules), then a *cannot* cross this membrane in the opposite direction. These systems are considered in Section 8.7. Unary rules are a special case of non-cooperative rules,

i.e., rules that describe the passage of a single object (which during the passage can be changed into a multiset of objects). CAS P systems using only non-cooperative rules are considered in Section 8.8 both with and without the use of antiport rules. In Section 8.9 we consider *bounded* CAS P systems, i.e., CAS P systems for which there exists a positive integer k such that in any computation and any region the cardinality of the multiset of objects present in the region does not exceed k. In the last section we discuss the results obtained in this chapter and we formulate a number of open problems.

8.2 Preliminaries

We assume the reader to be familiar with basic notions of formal languages and automata theory (which can be found, e.g., in [25]). In this section we briefly recall some notions and results concerning matrix grammars and register machines that will be used in some proofs in this chapter.

8.2.1 Matrix Grammars

A *matrix grammar* is a construct $G = (N, T, S, M, F)$, where N is the nonterminal alphabet, T is the terminal alphabet ($N \cap T = \varnothing$), $S \in N$ is the axiom, M is a finite set of sequences (called matrices) of context-free productions $(A_1 \to x_1, \ldots, A_n \to x_n), n \geq 1$ with $A_i \in N, x_i \in (N \cup T)^*, 1 \leq i \leq n$, and F is a set of occurrences of rules in M (note that one may have the same rule in different entries of a matrix and only some of these entries in F).

Given two strings $w \in (N \cup T)^* N (N \cup T)^*$ and $z \in (N \cup T)^*$, we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n) \in M$ and there exist strings $w_i \in (N \cup T)^*$, for $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$ and, for each $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i$ and $w_{i+1} = w'_i x_i w''_i$ for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and (the given occurrence of) production $A_i \to x_i$ appears in F. Thus, the productions of a matrix are applied in the order in which they are listed, except that one skips the rules in F if they cannot be applied – therefore we say that these productions are applied in the *appearance checking mode*. If the set F is empty, then G is said to be *without appearance checking*. The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages generated by matrix grammars with appearance checking is denoted by MAT_{ac}. The family of languages generated by matrix grammars without appearance checking is denoted by MAT. It is known (see [24, Chapter 12]) that $MAT \subset MAT_{ac} = RE$.

We say that a matrix grammar is *pure* if there is no distinction between terminals and nonterminals, i.e., each string derived from S belongs to $L(G)$. The family of languages generated by pure matrix grammars without appearance checking is denoted by $pMAT$. It is known (see [11, Lemma 5.1.1]) that $pMAT \subset MAT$.

8.2.2 Register Machines

Intuitively, a register machine is an automaton with a number of registers (each storing a natural number) that is executing labelled instructions of several simple types. More precisely:

A *register machine* is a construct $M = (k, \mathcal{P}, l_0, l_h)$, where:

- k is the number of registers,
- $\mathcal{P}$ is a set of labelled instructions (the *program*) that can be of the following forms:
  1. $(l : add(r), l_i, l_j)$,
  2. $(l : sub(r), l_i, l_j)$,
  3. $(l_h : halt)$,

  with l, l_h, l_i, l_j from the set $lab(\mathcal{P})$ of labels associated with the instructions in a one-to-one manner,
- $l_0 \in lab(\mathcal{P})$ is the label of the initial instruction,
- $l_h \in lab(\mathcal{P})$ is the label of the halting instruction.

The execution of an instruction $(l : add(r), l_i, l_j)$ increments by one the value stored in register r and then the machine proceeds, in a nondeterministic way, either to the instruction with label l_i or to the instruction with label l_j. An instruction $(l : sub(r), l_i, l_j)$ is executed as follows. If the value stored in register r is not zero, then it subtracts one from this value, and the machine proceeds to the instruction labelled by l_i; otherwise it proceeds to the instruction labelled by l_j. A halting instruction $(l_h : halt)$ stops the machine; its label is always the final label l_h.

We say that a vector $(n_1, \ldots, n_\alpha) \in \mathbb{N}^\alpha$ *is generated* by M (where α is fixed for M) if, starting from the instruction labelled by l_0 with the value of all registers equal to zero, it halts with value n_j in register j for all $1 \leq j \leq \alpha$, and with the values of the registers $\alpha + 1, \ldots, k$ equal to zero. The set of all vectors obtained in this way constitutes the set generated by M. The family of all sets of vectors generated by register machines is denoted by $RegM$. It is known (see [19]) that register machines can generate the family of Turing computable sets of vectors of natural numbers, that is, $RegM = PsRE$.

A *register machine without checking for zero* is a register machine where all subtraction instructions are of the form $(l : sub(r), l_j, l_{h^*})$, where l_{h^*} is the non-successful halting label – the machine stops but the computation is not considered successful. The family of languages generated by register machines without checking for zero is denoted by $RegM_{\neq 0}$. The computational power of this model is the same, in terms of Parikh images of languages, as that of matrix grammars without appearance checking. That is, $RegM_{\neq 0} = PsMAT$ (see, e.g., [12]).

8.3 Communication Membrane Systems with Active Symports

In what follows we assume that the reader is familiar with the area of membrane computing, in particular with membrane systems using symport/antiport rules, see, e.g., [21]. We will still use a rather informal terminology describing the nested relationship between membranes or regions, such as, e.g., (immediately) inner membrane of membrane i or (immediately) outer membrane of membrane i, however, if needed, this can be always made precise by, e.g., considering a tree representation of the nested structure of membranes (thus an immediately inner membrane of membrane i becomes a *direct descendent* of membrane i).

The model of membrane systems studied in this chapter is defined as follows.

Definition 1
A *communication membrane system with active symports* (in short, a *CAS P system*) is a construct

$$\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, R_1^a, \ldots, R_m^a, i_0),$$

where:

- Γ is the alphabet of objects,
- μ is a tree structure representing a membrane structure with m membranes (labelled in a one-to-one manner by $1, \ldots, m$),
- $w_1, \ldots, w_m$ are the multisets of objects initially present in the regions of the system,
- $R_1, \ldots, R_m$ are finite sets of symport-rewriting (in short, sr) rules associated with membranes $1, 2, \ldots, m$, respectively; each rule is of the form (u, v, out) or (u, v, in), where $u \in \Gamma^+, v \in \Gamma^*$,
- $R_1^a, \ldots, R_m^a$ are finite sets of antiport rules associated with membranes $1, 2, \ldots, m$, respectively; each rule is of the form $(u, in; v, out)$ where $u \in \Gamma^+, v \in \Gamma^+$,
- $i_0 \in \{1, \ldots, m\}$ is the label of the output membrane of Π. ∎

We also use Γ_Π to denote Γ. As usual, the root of μ is called the *skin membrane*. The skin membrane separates the system from the *environment*. The leaves of μ are called *elementary*. Each membrane i delimits a *region* i: it is the space between it and its direct descendants. The *surrounding region* of membrane i is the environment if i is the skin membrane, and otherwise it is the region of the direct ancestor of i.

We now define the semantics of symport-rewriting rules. An sr rule $r = (u, v, out) \in R_i$, $1 \leq i \leq m$, can only be applied if multiset u is present in

region i (we say then that rule r is *applicable*). The effect of applying this rule is as follows: multiset u is deleted from region i and simultaneously multiset v is added to its surrounding region. This formalizes the following intuition: multiset u is moved *out of* region i by crossing through membrane i and *while* crossing, it is transformed into multiset v. An sr rule (u, v, in) from the set R_i is applied analogously, however multiset u is moved *into* region i by crossing through membrane i and while crossing u is transformed into multiset v. Rules $(u, v, out) \in R_i$ and $(u, v, in) \in R_i$ will also be denoted by $u \xrightarrow{\ ]_i\ } v$ and $u \xrightarrow{\ [_i\ } v$, respectively.

The (standard) effect of applying an *antiport rule* $(x, in; y, out) \in R_i^a$, $1 \leq i \leq m$, is as follows: multiset x crosses membrane i from the surrounding region into region i, while, at the same time, multiset y moves in the opposite direction through membrane i.

Before going on, it is worth to remark that CAS P systems do not assume a potentially infinite supply of objects available in the environment, although this is the case in the classical definition of symport/antiport P systems, which have a set E of objects available in the environment in an unbounded number of copies.

In this chapter we consider several restrictions for the sr rules of CAS P systems, and so we give now notation and terminology to describe these restrictions. Symport-rewriting rules (u, v, out) and (u, v, in) are called *cooperative* if $|u| \geq 2$, and *noncooperative* if $|u| = 1$. Also, sr rules are called *alph-restricted* if $alph(v) \subseteq alph(u)$, where $alph(x)$ is the smallest alphabet $\Psi \subseteq \Gamma$ such that $x \in \Psi^*$. Thus, such sr rules cannot introduce new types of objects. Noncooperative alph-restricted sr rules are also called *unary* sr rules (because there is only one type of object, one symbol, present in these rules). The *weight* of an antiport rule $(u, in; v, out)$ is defined as $\max\{|u|, |v|\}$.

As usual, a *configuration* of a membrane system is an instantaneous description of the membrane structure and the contents of all the regions. The *initial configuration* consists of the membrane structure μ and the multisets of objects initially present in the regions of the system, given by $w_1, \ldots, w_m$.

The system evolves from one configuration to another by performing a *transition step*. In one mode of operation, the most usual one for P systems, the transition steps are performed by applying the rules in a nondeterministic, maximally parallel manner. However, we will also consider another mode of operation, called *sequential*, where no antiport rules are present and *at most one* sr rule is applied at each step for each membrane, allowing a membrane to be inactive even when there is an applicable sr rule.

All the possible sequences (finite or infinite) of transition steps that the system is able to perform from the initial configuration are the *computations* of the system. A given configuration is called *reachable* if it results from a computation of the system. A reachable configuration is a *halting configuration*, if there is no rule applicable to it. The *output* of a CAS P system, denoted by $Ps(\Pi)$, is the set of vectors of natural numbers which are the Parikh images of the multisets present in region i_0 in all possible halting configurations.

We also want to make a comment concerning the outputs of computations in

membrane systems. They are traditionally either vectors expressing the multiplicities of various objects present in the output region at the conclusion of a successful computation or numbers expressing the cardinality of *all* objects present in the output region at the conclusion of a successful computation. The common dimension of the vectors is the cardinality of a fixed a priori alphabet of output objects. This alphabet may be different from the total alphabet Γ of objects, however it is not *explicitly* given. This also happens in this chapter, however as usual the output alphabet is always well understood from the context of considerations.

Example 1
It is easy to see that the (unique) halting configuration of the CAS P system Π from Figure 8.1 is (in both sequential and maximally parallel mode) $[\,[\,]_2\,[\,aaaaaaaa]_3\,]_1$ and so, we have $Ps(\Pi) = \{(8)\}$.

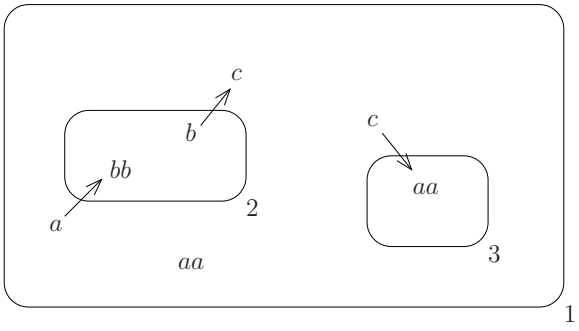

Figure 8.1: A graphic representation of a communication membrane system Π with active symports (CAS P system). The initial configuration is $[\,[\,]_2\,[\,]_3\,a\,a]_1$. The output region is the one enclosed by membrane 3 and the output alphabet is $\{a\}$. The sets of sr rules associated with regions 2 and 3 are: $R_2 = \{b \rightarrow_{\lfloor 2} c,\ a \rightarrow_{\lfloor 2} bb\}$ and $R_3 = \{c \rightarrow_{\lfloor 3} aa\}$, respectively.

We denote by $\mathrm{CAS}(\alpha, ant_i)$ the class of CAS P systems with sr rules of type α, and antiports of weight at most i. In this chapter we consider $\alpha \in \{coo, ncoo, coo_{AR}, ncoo_U\}$ to denote general (non-restricted), noncooperative, cooperative alph-restricted, and unary sr rules, respectively. We use $PsCAS_m(\alpha, ant_i)$ to denote the family of sets of vectors computed by CAS P systems from the class $\mathrm{CAS}(\alpha, ant_i)$ with at most m membranes. Besides, we denote $PsCAS_*(\alpha, ant_i) = \bigcup_{m\in\mathbb{N}} PsCAS_m(\alpha, ant_i)$. Moreover, when we consider systems working in the sequential mode we will add the prefix *seq* (obtaining in this way the notation $seqPsCAS_m(\alpha, ant_i)$).

In what follows, to simplify the notation, in the definition of CAS P systems that do not use antiport rules we will omit the specification of the sets of antiport rules.

8.4 Alphabetic Restriction

In this section we investigate systems using *cooperative alph-restricted* sr rules, coo_{AR} sr rules for short, *without* antiport rules. We show that this restriction does not decrease the generative power of the model – CAS P systems using coo_{AR} sr rules are computationally universal.

Theorem 2
$PsCAS_2(coo_{AR}, ant_0) = PsRE$.

Proof
In order to demonstrate the computational universality of the model, we will prove that any register machine can be simulated by it. To this aim, consider an arbitrary register machine $M = (k, \mathcal{P}, l_0, l_h)$, generating a set of vectors over $\mathbb{N}^\alpha$ (i.e., registers from 1 to α are *output registers* and registers from $\alpha + 1$ to k are *working registers*), where $lab(\mathcal{P})$ consists of n labels. We construct the CAS P system $\Pi = (\Gamma, \mu, w_1, w_2, R_1, R_2, i_0)$ satisfying the required conditions that generates the same set of vectors as M does, as follows.

- $\Gamma = \{c_i \mid 1 \leq i \leq k\} \cup \{l_i, l_i', l_i'', l_i''' \mid 0 \leq i \leq n-1\}$.
- $\mu = [\ [\ ]_2\ ]_1$.
- $w_1 = \lambda$ and $w_2 = l_0 c_1 \dots c_k l_0 l_0' l_0'' l_0''' \dots l_{n-1} l_{n-1}' l_{n-1}'' l_{n-1}'''$.
- $R_1 = \varnothing$,
- The set R_2 of cooperative alph-restricted sr rules is defined as follows:
  - for every instruction $\big(l_{i_1} : add(j), l_{i_2}, l_{i_3}\big) \in \mathcal{P}$, R_2 contains the sr rules $l_{i_1} l_{i_1} l_{i_2} c_j \dashrightarrow_{]_2} l_{i_2} l_{i_2} l_{i_1} c_j c_j$ and $l_{i_1} l_{i_1} l_{i_3} c_j \dashrightarrow_{]_2} l_{i_3} l_{i_3} l_{i_1} c_j c_j$,
  - for every instruction $rs = \big(l_{i_1} : sub(j), l_{i_2}, l_{i_3}\big) \in \mathcal{P}$, R_2 contains the sr rules
    $rs_1 = l_{i_1} l_{i_1} l_{i_1}' l_{i_1}'' \dashrightarrow_{]_2} l_{i_1} l_{i_1}' l_{i_1}' l_{i_1}'' l_{i_1}'' l_{i_1}''$,
    $rs_2 = l_{i_1}' l_{i_1}' l_{i_1}'' \dashrightarrow_{]_2} l_{i_1}' l_{i_1}'' l_{i_1}''$,
    $rs_3 = l_{i_1}'' l_{i_1}'' l_{i_1}''' c_j c_j \dashrightarrow_{]_2} l_{i_1}'' l_{i_1}''' l_{i_1}''' c_j$,
    $rs_4 = l_{i_1}'' l_{i_1}'' l_{i_1}'' l_{i_1}''' l_{i_1}''' l_{i_2} \dashrightarrow_{]_2} l_{i_2} l_{i_2} l_{i_1}'' l_{i_1}'''$,
    $rs_5 = l_{i_1}'' l_{i_1}'' l_{i_1}'' l_{i_1}'' l_{i_3} \dashrightarrow_{]_2} l_{i_3} l_{i_3} l_{i_1}''$.
  - in addition, R_2 contains the following sr rules:
    $x \dashrightarrow_{]_2} x$, for every $x \in \Gamma$, and
    $rs_6 = l_h c_1 \cdots c_k l_0 l_0' l_0'' l_0''' \cdots l_{n-1} l_{n-1}' l_{n-1}'' l_{n-1}''' \dashrightarrow_{]_2} \lambda$.
- $i_0 = 2$.

The simulation of M by Π is performed as follows. The current label of M is represented by exactly one symbol l_i that appears in region 2 with multiplicity two. Therefore, the initial label l_0 of M appears twice in w_2. The number n_i stored in a register i of M is encoded in Π by multiplicity $n_i + 1$ of symbol c_i. The

reason for these encodings is that since only coo_{AR} sr rules are available, during the computation, the multiplicity of c_i and l_i may never be zero. Thus, only one copy of c_j corresponds to the case that register j is *zero*. Since in the initial configuration of M every register is zero, each symbol c_i appears in multiplicity one in w_2.

The interpretation of the sr rules that simulate the addition instruction $\left(l_{i_1} : add(j), l_{i_2}, l_{i_3}\right) \in \mathcal{P}$ is quite straightforward: we modify the amount of objects l_{i_1} and l_{i_2} (or l_{i_3}) to simulate the transfer of control from l_{i_1} to l_{i_2} (or to l_{i_3}, respectively). We also generate one more copy of object c_j to simulate the increment by 1 of the value of register j. Note that by applying this sr rule we move objects from region 2 to region 1. In the next step all the objects are put back to region 2 without modifying their multiplicities. This is accomplished by sr rules $x \longrightarrow\!\!\!\!\!\!\vdash_{\lfloor 2} x$, for every $x \in \Gamma$.

The simulation of the subtraction instruction is more complicated since we need to check whether or not the value stored in the specified register is zero in order to select the next control label of the machine. Given $rs = \left(l_{i_1} : sub(j), l_{i_2}, l_{i_3}\right) \in \mathcal{P}$, we proceed in three steps (as before, after each of these steps all objects are put back into region 2) depending on whether or not the value of register j in $\mathcal{P}$ is zero (or, equivalently, on whether or not the multiplicity of c_j in Π is one):

step	register j is not zero	register j is zero
1	rs_1	rs_1
2	rs_2, rs_3	rs_2
3	rs_4	rs_5

The end result is that the multiplicity of l_{i_1} is changed from two to one, and the multiplicity of l_{i_2} (l_{i_3}) is changed from one to two when the value stored in register j was not zero (was zero, resp.). Moreover, when register j was not zero, the multiplicity of c_j was decreased by one.

Thus, Π simulates M step by step until arriving at the label l_h (in case of a successful computation). Then, Π performs one last step deleting all auxiliary objects through the application of sr rule rs_6. Thus, in the halting configuration region 1 will be empty, and the multiplicity of objects c_j, $1 \leq j \leq \alpha$, in region 2 represents exactly the contents of the corresponding output registers of the machine in the halting state. Consequently, since $RegM = PsRE$ we have proved that $PsCAS_2(coo_{AR}, ant_0) \supseteq PsRE$. Invoking the Turing-Church thesis we obtain also the converse inclusion; hence $PsCAS_2(coo_{AR}, ant_0) = PsRE$. ∎

Note that the maximal parallelism of the system is fundamental in the above proof, as it allows to simulate the zero checking in a register. Indeed, rule rs_3 is applicable in step 2 iff the value stored in the corresponding register is not zero.

Remark

The above proof can be modified in order to decrease the cardinality of the alphabet of Π. Actually, all the elements in the set $\Lambda = \{l_i, l_i', l_i'', l_i''' \mid i = 0, \ldots, n-1\}$ can be encoded by using only two symbols. The idea is to establish a correspondence between elements from Λ and pairs of numbers, each pair being represented

by the multiplicities of two symbols (e.g., a and b). This must be done in such a way that if $l_1 \in \Lambda$ is represented by a smaller multiplicity of a than used for representing l_2, then the multiplicity of b in the representation of l_1 must be greater than the multiplicity of b in the representation of l_2. In this way, a given multiplicity of a and b (encoding $l_1 \in \Lambda$), can trigger only l_1.

Corollary 3
$PsCAS_2(coo, ant_0) = PsRE$.

Proof
Since the set of CAS P systems using cooperative sr rules includes the set of CAS P systems using coo_{AR} sr rules, we have $PsCAS_2(coo_{AR}, ant_0) \subseteq PsCAS_2(coo, ant_0)$. Thus, the corollary follows from the previous theorem and the Turing-Church thesis. ∎

8.5 The Sequential Mode

The application of sr rules in a maximally parallel way is a powerful way to regulate the communication. This fact will become even more clear in this section where we consider CAS P systems working in the sequential mode: in each membrane *at most one* sr rule is applied at each step, allowing a membrane to remain inactive even when there is an applicable sr rule for it. The notion of a sequential mode of operation for P systems with carriers is studied in [15], and the computational power of these systems is shown to be equal to the family of Parikh images of the languages generated by matrix grammars without appearance checking. We show that the generative power of CAS P systems with cooperative sr rules working in the sequential mode is also equal to $PsMAT$. First, we prove the result for cooperative alph-restricted sr rules, and later we extend it to cooperative sr rules.

Lemma 4
$seqPsCAS_*(coo_{AR}, ant_0) \subseteq PsMAT$.

Proof
Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$ be a CAS system with cooperative alph-restricted sr rules and working in the sequential mode. We prove that there exists a matrix grammar without appearance checking generating a language whose Parikh image is $Ps(\Pi)$.

First of all, as matrix grammars can only handle strings, we need to find a string-representation for the configurations of Π (which indicates for each object the region in which it occurs). To this aim, we introduce a notation that assigns to each object a pair (*object*, *location*) explicitly mentioning the region where the object resides. That is, for each object x present in region i, we include the pair (x, i) in the string. Note that the so-constructed string may contain many occurrences of a pair (x, i) – this represents the multiplicity of x in region i.

Note also that the order of such pairs in a string is not relevant, so any permutation of the string could be used as well. We will use *pure matrix grammars* to generate every reachable configuration of Π, and later filter out the halting configurations.

In order to correctly simulate the transition steps in Π by means of matrices of context-free productions, we have to avoid that the pairs (x, i) corresponding to objects produced in a given step are used to trigger rules in the *same* transition step. Furthermore, we have to take into account all the sr rules that are applied in Π (recall that Π works in the sequential mode, so at most one sr rule is applied in each membrane at each step). We will introduce one matrix for each possible applicable multiset of sr rules. That is, we will have a specific matrix to simulate each one of the $(|R_1|+1)\cdot(|R_2|+1)\cdots(|R_m|+1)$ possible combinations of selecting at most one sr rule from each one of the sets of sr rules associated with the m membranes of Π.

We are ready now to specify the pure matrix grammar without appearance checking $G_\Pi = (N, S, M)$ that simulates Π. Let $N = (\Gamma\times\{1,\ldots,m\})\cup\{S\}\cup\{A_{j,i} \mid 1 \leq i \leq m,\ 1 \leq j \leq |R_i|\}$ and $S \notin \Gamma$. The set of matrices M is defined as follows. Let $R_i = \{r_1^i, \ldots, r_{n_i}^i\}$, for $1 \leq i \leq m$, and consider an arbitrary set C consisting of $c \leq m$ sr rules taken from $R_1, \ldots, R_m$ (at most one from each set), $C = \{r_{j_1}^{i_1}, \cdots, r_{j_c}^{i_c}\}$. For each sr rule $r_j^i : u \rightarrow\!\!\!\!\!\!\vdash_i v$ in C we define

$$avail(r_j^i) = \{(x_1, i) \to A_{j,i};\ (x_2, i) \to \lambda;\ \ldots\ ;\ (x_k, i) \to \lambda\},$$
$$prod(r_j^i) = \{A_{j,i} \to (y_1, i')(y_2, i')\cdots(y_{k'}, i')\},$$

where $u = x_1 \ldots x_k$, $v = y_1 \ldots y_{k'}$, and i' is the surrounding region of i (we proceed analogously for sr rules $u \rightarrow\!\!\!\!\!\!\vdash_i v$, by just exchanging i and i'). Then, for each C as above, we let M contain the matrix

$$\big(avail(r_{j_1}^{i_1}); \ldots; avail(r_{j_c}^{i_c}); prod(r_{j_1}^{i_1}); \ldots; prod(r_{j_c}^{i_c})\big)$$

that simulates the application of rules in C. The set M also includes a matrix $(S \to w_0)$, where w_0 is a string-representation of the initial configuration of Π.

Thus, we simulate Π by G_Π, in such a way that the language generated by G_Π corresponds to the string-representation of the reachable configurations of Π. Now, taking into account the inclusion $pMAT \subset MAT$ (cf. Section 8.2), we know that there exists a matrix grammar without appearance checking G such that $L(G) = L(G_\Pi)$.

Next, we need to filter out words from $L(G)$ corresponding to the halting configurations of Π. Note that the number of sr rules of Π is finite, and each one of them has a finite number of symbols on its left hand side. Therefore, the set of all configurations to which no rules of Π are applicable forms a regular language. In this way, it is possible to obtain the halting configurations of Π as the intersection of $L(G)$ and a regular language. Since the family of languages generated by matrix grammars without appearance checking is closed under intersection with regular languages (see [11], Lemma 1.3.5), we deduce that there exists a ma-

trix grammar without appearance checking G' such that $L(G')$ contains all the string-representations of the halting configurations of Π.

Finally, we apply the following erasing morphism over the set of halting configurations:

$$\gamma((x,i)) = \begin{cases} \lambda & \text{if } i \neq i_0, \\ x & \text{if } i = i_0, \end{cases}$$

and we get exactly the contents of the output region for every halting configuration, the Parikh image of which are the vectors generated by Π. Thus, taking into account that MAT is closed under morphisms (see [11], Theorem 1.3.1), we conclude that there exists a matrix grammar without appearance checking G'' that generates, in the Parikh image sense, the same set of vectors as Π. Therefore, we have that $seqPsCAS_*(coo_{AR}, ant_0) \subseteq PsMAT$. ∎

Lemma 5
$PsMAT \subseteq seqPsCAS_2(coo_{AR}, ant_0)$.

Proof
Since $RegM_{\neq 0} = PsMAT$ (see Section 8.2) we prove that any register machine without checking for zero can be simulated by a CAS P system.

Consider an arbitrary register machine $M = (k, \mathcal{P}, l_0, l_h)$ with k registers and without checking for zero, generating a set of vectors over $\mathbb{N}^n$ (that is, registers from 1 to n are *output registers* and registers from $n+1$ to k are *working registers*). We define the CAS P system $\Pi = (\Gamma, \mu, w_1, w_2, R_1, R_2, i_0)$ as follows:

- $\Gamma = \{c_1, \ldots, c_k, l, l_0, \ldots, l_{n-1}\}$,
- $\mu = [\ [\ ]_2\ ]_1$,
- $w_1 = \lambda$, $w_2 = l_0 l l_0 l_1 \ldots l_{n-1} c_1 \ldots c_k$,
- $R_1 = \varnothing$,
- the set R_2 of cooperative alph-restricted sr rules is defined as follows:
  1. for every instruction $(l_{i_1} : add(j), l_{i_2}, l_{i_3}) \in \mathcal{P}$, R_2 contains the following sr rules: $l_{i_1} l_{i_1} l_{i_2} c_j \rightarrow_{]_2} l_{i_1} l_{i_2} l_{i_2} c_j c_j$,
     $l_{i_1} l_{i_2} l_{i_2} c_j c_j \rightarrow_{[_2} l_{i_1} l_{i_2} l_{i_2} c_j c_j$,
     $l_{i_1} l_{i_1} l_{i_3} c_j \rightarrow_{]_2} l_{i_1} l_{i_3} l_{i_3} c_j c_j$,
     $l_{i_1} l_{i_3} l_{i_3} c_j c_j \rightarrow_{[_2} l_{i_1} l_{i_3} l_{i_3} c_j c_j$.
  2. for every instruction $(l_{i_1} : sub(j), l_{i_2}, l_{h^*}) \in \mathcal{P}$, R_2 contains the following sr rules: $l_{i_1} l_{i_1} l_{i_2} c_j c_j \rightarrow_{]_2} l_{i_1} l_{i_2} l_{i_2} c_j$ and $l_{i_1} l_{i_2} l_{i_2} c_j \rightarrow_{[_2} l_{i_1} l_{i_2} l_{i_2} c_j$.
  3. in addition, R_2 contains the following sr rules:
     $c_j \rightarrow_{]_2} c_j$ and $c_j \rightarrow_{[_2} c_j$ for every $j \in \{n+1, \ldots, k\}$,
     $l \rightarrow_{]_2} l, l \rightarrow_{[_2} l$,
     $c_1 \ldots c_k l_h l l_0 l_1 \ldots l_{n-1} \rightarrow_{]_2} \lambda$.
- $i_0 = 2$.

Since we again deal with coo_{AR} sr rules, we use the same encodings for the contents of the registers and for the current label of M as we did in the proof of Theorem 2. The current label of M is represented by exactly one symbol l_i that appears in region 2 with multiplicity two (the others have multiplicity one), and the number n_i stored in a register i of M is encoded in Π by multiplicity $n_i + 1$ of symbol c_i.

The intuition behind the construction of Π is as follows. The addition instruction is simulated by just generating one more object for the corresponding register and replacing a copy of l_{i_1} nondeterministically by a copy of either l_{i_2} or l_{i_3}. For the subtraction instruction, if the register to which we try to apply the instruction is zero, then the rule $l_{i_1} l_{i_1} l_{i_2} c_j c_j \longrightarrow_{]_2} l_{i_1} l_{i_2} l_{i_2} c_j$ cannot be applied. Instead of getting a non-accepting halting label l_{h^*}, in Π we avoid halting and accepting in Π by using the infinite loop given by the execution of the rules $l \longrightarrow_{]_2} l$, $l \longrightarrow_{[_2} l$. The only case when the loop can be interrupted is when the corresponding computation of M is successful, that is, if the label l_h is reached. In this case, Π deletes l, together with the additional elements c_j for all the registers by applying the rule $c_1 \dots c_k l_h l l_0 l_1 \dots l_{n-1} \longrightarrow_{]_2} \lambda$. Then, we can get the output (the contents of the registers) from the multiset of objects present in region 2 in the halting configuration. Notice that if any of the working registers r_j with $j \in \{n+1, \dots, k\}$ is not zero when l_h is reached, then Π will not halt, because the corresponding rules $c_j \longrightarrow_{]_2} c_j$ and $c_j \longrightarrow_{[_2} c_j$ will run forever. ∎

Theorem 6
$seqPsCAS_2(coo_{AR}, ant_0) = seqPsCAS_*(coo_{AR}, ant_0) = PsMAT$.

Proof
By definition, the inclusion $seqPsCAS_2(coo_{AR}, ant_0) \subseteq seqPsCAS_*(coo_{AR}, ant_0)$ holds. By Lemma 4 and Lemma 5 we have the desired result. ∎

Corollary 7
$seqPsCAS_2(coo, ant_0) = seqPsCAS_*(coo, ant_0) = PsMAT$.

Proof
The proof follows easily from the previous theorem. On the one hand, the same construction of the matrix grammar used for proving Lemma 4 can be used for the general cooperative case. On the other hand, it is clear that $seqPsCAS_2(coo_{AR}, ant_0) \subseteq seqPsCAS_2(coo, ant_0)$. ∎

8.6 Unary Rules

We now consider unary sr rules instead of cooperative alph-restricted sr rules. Recall that unary sr rules are sr rules of the form $a \longrightarrow v$ with $v \in a^*$. Hence, one object a can move from one region to another and, while crossing the membrane, it can produce several copies of itself. It turns out that if we also allow standard antiport rules of weight one, then we get computational universality.

Theorem 8
$PsCAS_*(ncoo_U, ant_1) = PsRE$.

Proof
In order to prove the computational universality of the model, we rely on the proof in [26] which provides a construction of a universal P system Π with symport and antiport rules of weight 1, and with an environment containing an unbounded supply of objects.

We construct a CAS P system Π′ that simulates the computations of Π as follows. Since symport rules of weight 1 are a particular case of unary sr rules, we can include Π as a subsystem of Π′.

In the proof from [26], during the initial steps of a computation the system receives as an input arbitrarily many objects from the environment, but only one at each step. Thus, we simulate the "infinite environment" condition by using repetitive applications of rules that are able to generate an unbounded number of objects.

More precisely, we consider a new skin membrane surrounding Π that will play the role of the environment of Π, and two additional elementary membranes in the new skin region. Let e_0, e_1, and e_2 be the labels for these three membranes, respectively, and let $E = \{a_1, \ldots, a_n\}$ be the alphabet of the environment of Π (we refer again to [26]). Finally, we include in our system the initial multiset $w_{e_1} = a_1\bar{a}_1 \ldots a_n\bar{a}_n$, and we consider the following sets of rules:

- $R_{e_1} = \{a_i \longrightarrow_{e_1} a_i^2,\ \bar{a}_i \longrightarrow_{e_1} \bar{a}_i \mid 1 \leq i \leq n\}$.
- $R^a_{e_1} = \{(a_i, in; \bar{a}_i, out) \mid 1 \leq i \leq n\}$.
- $R_{e_2} = \{\bar{a}_i \longrightarrow_{e_2} \bar{a}_i \mid 1 \leq i \leq n\}$.

These rules behave as loops that are able to generate new objects a_i for every $1 \leq i \leq n$ as long as they continue (the loop for a given i halts when the object $\bar{a}_i$ gets inside membrane e_2). Therefore, such construction can produce nondeterministically arbitrarily many copies of objects a_i for $1 \leq i \leq n$ that will be used by the original system Π to implement the simulation described in the proof from [26].

We conclude that $PsRE \subseteq PsCAS_*(ncoo_U, ant_1)$, and since $PsCAS_*(ncoo_U, ant_1) \subseteq PsRE$ is assumed to be true, the theorem holds. ∎

Based on the above theorem we have the following result.

Corollary 9
$PsCAS_*(\alpha, ant_1) = PsRE$, for $\alpha \in \{ncoo_U, ncoo, coo_{AR}, coo\}$.

We conclude this section with a preliminary result concerning the generative power of systems from CAS$(ncoo_U, ant_0)$.

Theorem 10
$PsCAS_*(ncoo_U, ant_0)$ is incomparable with $PsFIN$.

Proof
On one hand it is clear that $PsCAS_*(ncoo_U, ant_0)$ contains an infinite set of vectors. Moreover $PsCAS_*(ncoo_U, ant_0)$ does not contain, for instance, the finite set $\{(2,3),(3,4)\}$. Every system from CAS$(ncoo_U, ant_0)$ that generates the vectors $(2,3)$ and $(3,4)$ can also generate the vectors $(2,4)$ and $(3,3)$ because the rules producing distinct symbols are noncooperative and unary, hence independent. ∎

8.7 Unidirectional Membranes

In this section we consider a constraint on the form of the rules for systems in CAS$(ncoo_U, ant_1)$, which strengthens the selective role of membranes in regulating the traffic of molecules through them. More specifically, this restriction makes the traffic strictly unidirectional: if a molecule can pass through a membrane in one direction, then it cannot pass through the same membrane in the opposite direction. Such a restriction is natural from the biological point of view: e.g., if a toxic substance is secreted from a cell through its membrane, then it should not be allowed to go back. This is interesting also from mathematical point of view, because it turns out to be a real restriction: we get a model that is not computationally universal.

This restriction is formally defined as follows. Let

$$\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, R_1^a, \ldots, R_m^a, i_0)$$

be a CAS$(ncoo, ant_1)$ P system. Let, for $i \in \{1, \ldots, m\}$,

$$\begin{aligned} In_i &= \{a \in \Gamma \mid (a, v, in) \in R_i, v \in \Gamma^* \text{ or } (a, in; b, out) \in R_i^a, b \in \Gamma\}, \\ Out_i &= \{a \in \Gamma \mid (a, v, out) \in R_i, v \in \Gamma^* \text{ or } (b, in; a, out) \in R_i^a, b \in \Gamma\}. \end{aligned}$$

Then Π is called *unidirectional* if In_i and Out_i are disjoint for all $i \in \{1, \ldots, m\}$. The class of all unidirectional systems in CAS$(ncoo, ant_1)$ is denoted uniCAS$(ncoo, ant_1)$. Moreover, for a set of sr rules R, we define $max(R) = \max\{|v| \mid (a, v, in) \in R \text{ or } (a, v, out) \in R\}$. For a configuration C we use $tob(C)$ to denote the total number of objects (thus with multiplicities counted) present in all the regions in C. The main consequence of unidirectionality is expressed in the following lemma.

Lemma 11
Let $\Pi \in uni$CAS$(ncoo_U, ant_1)$, let w be the multiset of objects initially present in Π, and let R be the set of all sr rules in Π. Then for every reachable configuration C in Π, $tob(C) \leq |w| \cdot (weight(R))^m$.

Proof
The unidirectionality of Π implies that no object in Π during any computation can cross the same membrane twice. Therefore, the upper bound on the total number of membranes crossed by an object in any computation is given by m. Since in one crossing each object cannot generate more than $weight(R)$ objects and $|w|$ is the number of objects in the initial configuration, $|w| \cdot (weight(R))^m$ is indeed an

upper bound on the total number of objects in every reachable configuration in Π. ∎

The next theorem shows that unidirectional P systems with unary sr rules and antiport rules of weight 1 generate only *finite* sets of vectors.

Theorem 12
$uniPsCAS_*(ncoo_U, ant_1) = PsFIN$.

Proof
Consider an arbitrary $\Pi \in uni\text{CAS}(ncoo_U, ant_1)$. By Lemma 11 the total number of objects in Π is bounded, and so the number of possible reachable configurations is finite. Therefore, $uniPsCAS_*(ncoo_U, ant_1)$ is also finite.

In order to prove the reverse inclusion, consider a finite set A of vectors of dimension n, and let $m = |A|$. We construct $\Pi_A \in uni\text{CAS}(ncoo_U, ant_1)$ that generates A as follows.

$$\Pi_A = (\Gamma, \mu, w_1, \ldots, w_{2m+2}, R_1, \ldots, R_{2m+2}, R_1^a, \ldots, R_{2m+2}^a, i_0),$$

where:

- $\Gamma = \{S, a_1, \ldots, a_n\}$;
- $\mu = [\,[\,[\,]_3\,]_2 \ \ldots\ [\,[\,]_{2m+1}\,]_{2m}\,[\,]_{2m+2}\,]_1$;
- $w_1 = S,\ w_{2m+2} = \lambda$, and $w_{2j} = \lambda,\ w_{2j+1} = a_1 \ldots a_n,$ for $1 \leq j \leq m$;
- The output membrane is $i_0 = 2m + 2$;
- $R_1 = \varnothing$, $R_{2m+2} = \{(a_i, a_i, in) \mid 1 \leq i \leq n\}$, and $R_1^a = R_{2m+2}^a = \varnothing$. For $1 \leq j \leq m$ we have
  $R_{2j} = \{(S, S^n, in)\} \cup \{(a_i, a_i^{x_{ji}}, out) \mid 1 \leq i \leq n,\ v_j = (x_{j1}, \ldots, x_{jn})\}$,
  $R_{2j}^a = \varnothing$,
  $R_{2j+1} = \varnothing$,
  $R_{2j+1}^a = \{(S, in; a_i, out) \mid 1 \leq i \leq n\}$.

The alphabet Γ consists of one symbol a_i for each component of the vectors, plus an additional starting symbol S. The membrane structure is set up as follows. For each vector $v_j \in A$ we have two membranes (labelled by $2j$ and $2j+1$) in the membrane structure, and we also have another membrane to collect the output. The initial configuration contains object S in the skin, and the multiset $a_1 \ldots a_n$ placed in all the regions delimited by elementary membranes (except for region $2m + 2$).

The computation in Π_A proceeds as follows. In the first step, the object S enters a nondeterministically chosen membrane labelled by $2j$, $1 \leq j \leq m$, and it produces n copies of itself. Then, objects $a_1, \ldots, a_n$ present in region $2j + 1$ are exchanged via antiport rules against the n copies of S. In the next step, all those objects a_i cross membrane $2j$ towards the skin region, producing several copies

of themselves, according to the components of vector v_j. Finally, all these objects go to the output region (via sr rules of weight 1), and the computation stops. Consequently, we have $PsFIN \subseteq uniPsCAS_*(ncoo_U, ant_1)$, and so the theorem follows. ∎

The unidirectional restriction has the intuitive meaning of letting each object cross a membrane only in one direction. Note that it is easy to overcome this restriction in the case when the objects can be rewritten. Indeed, we will see in the next section that the unidirectional restriction does not decrease the power of the model if the sr rules are not unary.

8.8 Noncooperative Rules

We turn now to CAS P systems using noncooperative sr rules (without antiport rules). We will prove that such systems are not universal. In this proof we will use the class of P systems with symbol rewriting noncooperative rules using targets [21], which we denote by $OP(ncoo)$. Let $PsOP(ncoo)$ be the family of sets of vectors generated by the P systems in the class $OP(ncoo)$.

Theorem 13
$uniPsCAS_2(ncoo, ant_0) = PsCAS_*(ncoo, ant_0) = PsCF$.

Proof
First of all we recall (from [21]) that $PsOP(ncoo) = PsCF$. Using this result we need to prove two more inclusions in order to demonstrate the theorem.

1. $PsCAS_*(ncoo, ant_0) \subseteq PsOP(ncoo)$.

   This inclusion holds because any $\Pi \in \text{CAS}(ncoo, ant_0)$ can be reformulated as a P system $\Pi' \in OP(ncoo)$ with symbol-objects and noncooperative rules using targets in such a way that $Ps(\Pi') = Ps(\Pi)$. This reformulation is done as follows.

   The system Π' has the same alphabet, membrane structure and output region as Π. The sr rules $a \to_i v$ from Π are reformulated as $a \to (y_1, out)(y_2, out) \dots (y_k, out) \in R_i$ in Π' where $v = y_1 \dots y_k$. Analogously, sr rules $a \to v$ from Π are reformulated as rules $a \to (y_1, in_i)(y_2, in_i) \dots (y_k, in_i) \in R_j$ in Π', where j is the surrounding region of i.

   Obviously, $Ps(\Pi') = Ps(\Pi)$, and hence the inclusion holds.

2. $PsOP(ncoo) \subseteq uniPsCAS_2(ncoo, ant_0)$.

   Let $\Pi = (\Gamma, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_0) \in OP(ncoo)$. Let then

$$\Pi' = (\Gamma', \mu', w_1', w_2', R_1', R_2', 2) \in CAS_2(ncoo, ant_0)$$

   be defined as follows

   - $\Gamma' = \Gamma \times \{1, \dots, m\}$.

- $\mu' = [\ [\ ]_2\ ]_1$.
- $w'_1 = \lambda$ and $w'_2 = h_1(w_1) \cdots h_m(w_m)$, where for $1 \leq i \leq m$, h_i is the morphism defined by $h_i(x) = (x, i)$ for $x \in \Gamma$.
- There are no sr rules associated with the skin ($R'_1 = \varnothing$).
- For each rule $r_t : a \to u \in R_i$ of Π, we include in R'_2 two sr rules:
  - $(a, i) \longrightarrow_{\rfloor 2'} A_t$,
  - $A_t \longrightarrow_{\lfloor 2'} v$,

  where v is obtained by adding (x, i) for each $(x, here) \in u$, adding (x, k) for each $(x, in_k) \in u$, and adding (x, j) for each $(x, out) \in u$ where j is the surrounding region of i.
- $i'_0 = 2$.

Note that in Π' we have collapsed the membrane hierarchy of Π: for each object x in Π a pair (x, i) is introduced which specifies the region i of object x.

The simulation of Π by Π' proceeds as follows. First, we encode into w'_2 the multisets $w_1, \ldots, w_m$ present in the initial configuration of Π using the morphisms h_i, $1 \leq i \leq m$. Then, each transition step of Π is simulated in Π' in two steps: a special object is sent out to region 1 for each rule from Π that has been triggered, and then this object returns into region 2 generating the products of the application of that rule in Π, taking into account the target indicators. This process is iterated until the system stops, and the output is collected in region 2 (the inner region). Consequently, we have that $PsOP(ncoo) \subseteq uniPsCAS_2(ncoo, ant_0)$. ∎

8.9 Deciding Boundness

We say that a membrane system is *bounded* if there exists a positive integer k such that in any region of any reachable configuration the cardinality of the multiset of objects present in the region does not exceed k.

It is easily seen that the boundness property is undecidable for arbitrary CAS P systems. In this section we show that it is decidable for CAS P systems using noncooperative sr rules.

In the proof of this result we will use the notion of dependency graph. This idea comes from classical formal language theory, but it has not been used in the P systems framework until recently (see [9]).

Definition 14
Let $\Pi = (\Gamma, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$ be a CAS P system with noncooperative sr rules. The *dependency graph* associated with it, g_Π, is defined as follows:

- Nodes: $\Gamma \times \{1, \ldots, m\}$ and a special node O.

- Edges: First, we include an edge $O \xrightarrow{p} (a,i)$ for every $a \in \Gamma$ and $1 \leq i \leq m$ such that $|w_i|_a = p$. In this way we include in the graph information about the initial configuration.
  Then, we add an edge $(a,i) \xrightarrow{p} (b,j)$ for any sr rule $a \longrightarrow_i u \in R_i$, where $|u|_b = p > 0$ and j is the surrounding region of i.
  Analogously, we add an edge $(a,j) \xrightarrow{p} (b,i)$ for any sr rule $a \longrightarrow_i u \in R_i$, where $|u|_b = p > 0$ and j is the surrounding region of i.
  We also add one edge from (a,j) to itself with weight 1 $(a,j) \xrightarrow{1} (a,j)$, if there is no sr rule $a \longrightarrow_j u$ and no sr rule $a \longrightarrow_i u$ for any i such that j is its surrounding region. ■

Clearly, the labels of the edges represent the multiplicity of the created objects. Since sr rules are applied in parallel, it is natural to define the weight of a path in g_Π as the multiplication of the labels of its edges.

Theorem 15
Let $\Pi \in \text{CAS}(ncoo, ant_0)$, and let $D \subseteq \Gamma_\Pi$. It is decidable whether or not there exists $k \in \mathbb{N}$ such that no region in any reachable configuration of Π contains more than k objects.

Proof
We present a constructive proof, describing an algorithmic procedure to decide if there exists such k for an arbitrary region ρ of Π. To this aim we will use the dependency graph associated with Π, denoted by g_Π.

We say that in a given dependency graph there is a "growing loop" for objects from D in region ρ if there exists a circular path in g_Π such that:

1. at least one of its edges has a label greater than 1,
2. at least one node of the form (a,ρ) with $a \in D$ is reachable from any of the nodes in the circular path,
3. at least one node of such a circular path is reachable from the special node O of g_Π.

The algorithm consists of checking if there exists a "growing loop" that is able to eventually produce an unbounded number of objects from D in region ρ. Since g_Π is finite, this can be decided.

If such a loop exists, then we conclude that the number of objects from D in ρ is *not* bounded.

Conversely, if such a loop does not exist, then there exists an upper bound on the number of objects from D in region ρ.

Thus, applying the above described algorithm for all the regions of Π we can decide the existence of a bound on the number of objects. ■

Corollary 16
It is decidable whether or not an arbitrary $\Pi \in \text{CAS}(ncoo, ant_0)$ is bounded.

Proof
The proof follows from Theorem 15, by simply setting the entire alphabet of Π as the set D. ∎

In this section we provided a direct proof that the boundness property is decidable for CAS P systems with noncooperative sr rules. Note that one can also construct an indirect proof of this result using Theorem 13. Indeed, one can *construct* for each such CAS P system a context-free grammar generating the same language. Now using well known results from context-free grammars one obtains the boundness result.

8.10 Discussion

In standard membrane systems, the evolution (of objects) takes place in regions, and communication happens across membranes. In this chapter we have defined a class of membrane systems where membranes play a more central role: both communication and evolution are associated with membranes, and moreover evolution happens *only* as a result of communication.

We have presented an investigation of the *generative* power of CAS P systems. In particular, we have attempted to find out, in a systematic way, how various features/mechanisms of such systems influence their power.

First of all, we have shown that if there is some *context-sensitivity* in the system (that is, if we allow cooperative sr rules) then one obtains the computational universality, even if only cooperative alph-restricted sr rules (coo_{AR}) are used:

$$PsCAS_2(coo_{AR}, ant_0) = PsRE.$$

Also, allowing antiport rules makes the CAS P systems computationally universal, irrespectively of the type of sr rules used:

$$PsCAS_*(ncoo_U, ant_1) = PsRE.$$

Since antiports are rules where two objects residing in different sides of a membrane *cooperate* to exchange their positions, this supports the "context-sensitive" intuition of universality.

Thus "cooperation" is crucial for the control of communication in order to reach computational universality (this cooperation can be achieved by using antiport rules or by using cooperative sr rules).

We turned then to other ways of controlling communication. First, we studied the consequences of not allowing the parallel application of several sr rules on the same membrane (*seq* mode), and then we also explored the case when the communication channels are only unidirectional for each object. In both cases the computational universality is lost:

$$\begin{aligned} seqPsCAS_*(coo_{AR}, ant_0) &= PsMAT, \\ uniPsCAS_*(ncoo_U, ant_1) &= PsFIN. \end{aligned}$$

Finally, we have shown that computational universality is also lost if we explicitly required that the sr rules are noncooperative. This remains true even if we control the communication through membranes by unidirectionality:

$$\begin{aligned} PsCAS_*(ncoo, ant_0) &= PsCF, \\ uniPsCAS_2(ncoo, ant_0) &= PsCAS_*(ncoo, ant_0). \end{aligned}$$

These results are summarized in Figure 8.2. To avoid too cumbersome notation we omit here the subscripts indicating the number of membranes used. The identities involving coo_{AR} are also omitted, since they are equivalent to the *coo* case.

In order to get a better understanding of various features/mechanisms used in membrane systems, an even more systematic study is needed. Hence, e.g., one could consider "basic models" such as CAS P systems with unary sr rules (and without antiports), and determine their generating power.

It is somehow surprising to find out that $PsCAS(ncoo_U, ant_0)$ is incomparable with $PsFIN$.

Another interesting line of research is to study different ways of restricting the parallelism. In Section 8.7 we have studied membrane systems where *at most one* sr rule is applied on each membrane, but we allow that a membrane remains inactive even if there are applicable sr rules for it (*sequential mode*). If we impose the *maximality* condition to such sequential mode, which forces that *at least one* sr rule is applied on each membrane (provided that there are applicable rules for it), then we get some control over the application of the sr rules in the systems, and we again reach computational universality for the *cooperative* case. Thus, $max\text{-}seqPsCAS(coo, ant_0) \neq seqPsCAS(coo, ant_0)$. What is the situation in the noncooperative case?

Finally, it would be interesting to investigate the effect of minimal parallelism (see, [8]) in the framework of CAS P systems.

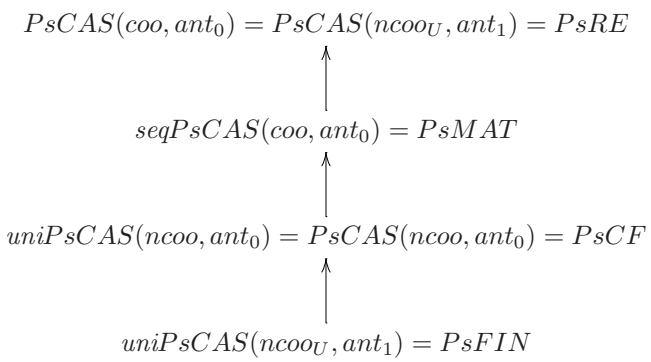

Figure 8.2: Hierarchy of families of sets of vectors computed by CAS P systems. Arrows indicate strict inclusions of the lower family in the upper family.